CMU-CS-84-113

Experience with the ZOG Human-Computer Interface System

Donald L. McCracken and Robert M. Akscyn

Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

February 1984

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

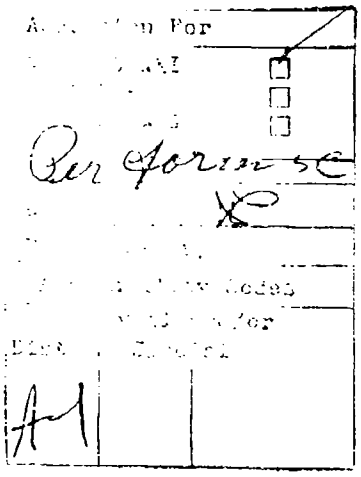
CMU-CS-84-113

Experience with the ZOG Human-Computer Interface System

Donald L. McCracken and Robert M. Akscyn

Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

February 1984



Abstract. This paper is primarily a reflection on more than eight years of research with the ZOG human-computer interface system. During that time we have experienced extensive use of ZOG. We begin the paper with a short description of the current ZOG implementation; then we proceed to a higher plane to describe a general ZOG philosophy that has evolved from our experience. Following the philosophy, we briefly describe the applications we have explored with ZOG, including a major application project for the Navy. Then we provide a critique of the current ZOG implementation by elucidating its strong and weak points. We end the paper with a brief glimpse at our plans for ZOG in the future.

This paper was presented at a workshop on Intelligent User Interfaces, 26-29 October, 1983, In Jackson, New Hampshire. The paper is also to appear in the July, 1984, issue of the *International Journal of Man-Machine Studies*.

Table of Contents

List of Figures

1. What is ZOG?

ZOG is a general-purpose, human-computer interface system based primarily on the concept of menu-selection, with a large database of menus and rapid response to selections (Robertson, G., McCracken & Newell, 1981). ZOG is intended to be used by both novice and expert users, providing a single interface mechanism that integrates all the computer functions needed by the user.

1.1. The structure of ZOG frames

The basic unit of representation in ZOG is called a *frame*. Originally, the notion of a frame meant a "structured screenful", i.e., everything the user could see on the terminal screen at one time. Nowadays, with the advent of high-resolution screens, implementations of ZOG provide for several frames to be displayed simultaneously on the screen. A ZOG frame consists of a set of items of different types, each of which carries its own positioning information. These item types are illustrated in the self-describing frame shown in Figure 1-1 below. A typical ZOG database may contain tens of thousands of interconnected frames.

```
This TITLE line summarizes the frame's contents                    IUI107

This TEXT expands the frame's main point of information, but is sometimes
omitted. The OPTIONS below are used to point to subordinate sections or to
provide an enumerated expansion of the main topic. LOCAL PADS do not have
the connotation of leading to deeper detail, but rather to tangential points
such as related material in another document or database. Invoking programs
is another function typically reserved for LOCAL PADS. At the bottom of the
frame is a set of general functions called GLOBAL PADS, which are made
available at every frame.

  1. This OPTION leads to another frame

  2. OPTIONS are often used like subpoints in an outline

  3.-This OPTION leads nowhere (indicated by the minus sign at the front)


                                A.-LOCAL PADS are used to point to
                                   peripheral information, or to
                                   invoke programs


edit help back next prev top goto acc mark ret zog disp user find info win xchg
```

Figure 1-1: A self-describing ZOG frame

1.2. Interaction with ZOG

There are three types of interactions with ZOG: *navigation*, *invoking programs*, and *editing*. The default mode of interaction is navigation, in which the user makes a selection via the keyboard or pointing device (mouse), and the system then responds by displaying the next frame. Most selections lead to other frames, but some have "actions", which perform a procedural function such as running a particular program. Finally, the user can enter the frame editor at any frame and make changes to the frame (if he has the requisite privileges, protection being implemented at the level of the individual frame).

To be concrete, suppose the user was faced with the frame shown in Figure 1-1 and wanted to know more. The user might select option 2, either by moving the mouse cursor to some part of option 2 and clicking a mouse button, or by typing the character 2 on the keyboard. Immediately, the frame would be replaced by the frame shown in Figure 1-2. The user might then want to go on, for example, by selecting option 1 of that frame. Alternatively, he might want to go back to the frame of Figure 1-1, which he could do by selecting the global pad **back**.

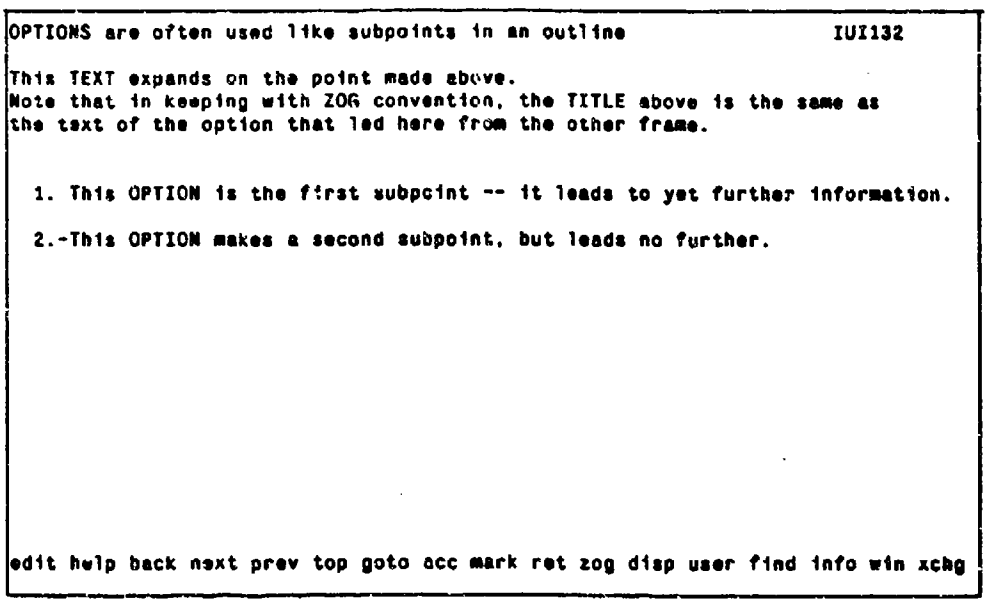


Figure 1-2: Second self-describing ZOG frame

1.3. The ZOG editor

The ZOG frame editor, ZED, is the principal editor for making changes to the database. It does more than a standard text editor, since it must deal with the substructure on the frame. For example, there are commands for changing the position of a selection, changing the type of a selection, adding a new selection, and changing what frame a selection links to.

1.4. Actions and agents

A selection in a ZOG frame can have an *action* associated with it. An action is a sequence of commands in the ZOG action language -- a simple programming language. This language contains commands for traversing the network, invoking intrinsic utilities, and entering the editor. However, the language also enables the user to invoke an arbitrary program. We call the programs *agents*, since typically their purpose is to perform some service on behalf of the user. In the current system, agents are simply Pascal programs that follow several conventions about how they receive input and return output, so that they can be integrated with the basic ZOG system. ZOG's functionality is extended by adding new agents.

1.5. History of ZOG

Work on ZOG at Carnegie-Mellon University (CMU) dates back to 1972. We built a precursor to the current ZOG system for participants in a cognitive science summer workshop, to allow them to easily use a wide variety of computer systems by providing a uniform interface. After the workshop, we shelved ZOG because 300 baud terminal technology was inadequate. We rekindled work on ZOG in 1975, after we became familiar with the PROMIS system at the University of Vermont -- a menu system based on rapid-response terminal technology, applied to the task of hospital management (Schultz & Davis, 1979). Our research goal was to understand in some generality the characteristics of large-network, rapid-response menu systems.

In 1980 we felt that ZOG was sufficiently mature to be tested in the real world. So we embarked on a major ZOG application project -- to build a computer-assisted management system for the Navy's newest nuclear-powered aircraft carrier, the USS CARL VINSON (Newell, McCracken, Robertson & Akscyn, 1981). This was a joint project between the ZOG Group at CMU and the officers of the CARL VINSON. It was unconventional because it attempted to transfer the results of university research directly into an operational environment. The development phase of the project ended in March, 1983, when the CARL VINSON left on her first deployment with a distributed ZOG system running on a network of 28 powerful personal computers (FERQs).

2. The Philosophy of ZOG

The ZOG system is based on a broad set of principles that we have come to call the *ZOG Philosophy*. This philosophy is a distillation of our experience with ZOG over the years. Some of the principles are general ones that we subscribed to before our work on ZOG, but most of them have evolved from our experience with ZOG. The principles provide a good description of the essential features of ZOG. They also can be interpreted as requirements that must be satisfied by an ideal ZOG implementation. We begin with some general philosophical tenets about the design of a human-computer interface, and then describe the three major components of the philosophy: the database, user interaction, and functional extension.

2.1. General tenets

Total environment. There should be a single environment with a single human-computer interface, in which the user can accomplish all of his various computing tasks. This avoids dealing with the idiosyncracies of many different user interfaces when switching from one program to another.

Flexible, efficient tool. The human-computer interface should be a flexible, efficient tool, not an active intelligence. Thus, there is no mystery about the workings of the interface -- it is under the total control of the user. Any intelligence the system may contain is in "frozen" form, i.e., in the content and structure of the viewable knowledge base. The place in such a system for active intelligence is at the *subsystem* level (e.g., as ZOG agents), not at the top level.

Direct manipulation of data. The user should be able to work with visual representations of data stored by the system, and be able to operate on that data incrementally with commands whose effect is immediately displayed (Shneiderman, 1983).

Semi-automatic operation. When a user needs to do something with the system that it is not specifically adapted for, it should be always possible to fall back to manual use of its general-purpose facilities. This gives the system a real robustness as it is pressed into new uses. When functions are used frequently, they can be automated by adding special purpose programs.

Low learning overhead. It should be easy to learn how to use the system, so that it can be used by people for whom the computer is only a tool to help them perform their real job. Many people are subject to such time pressure in their jobs that they cannot afford a large investment in learning to operate a computer system, even if the system would prove very helpful once learned.

Safe, exploratory environment. The system should provide an environment where it is safe to explore and learn by doing -- where there are no dangerous, irreversible actions that a user might stumble onto. The concept of an exploratory environment is described in some detail by Carroll (1982).

2.2. The database

A major component of a ZOG system is its database, which is somewhat different from the traditional notion of a database. The following principles govern the design of the database:

Large size. The database architecture should be able to accomodate hundreds of thousands of frames without adversely affecting the responsiveness of the system. This translates into a requirement for large secondary storage devices with a minimum capacity of around one billion bytes.

Shared by multiple users. The database should accomodate simultaneous use by many different users, so that it can provide a simple but rich means of communication among the users. Locking should be provided at the frame level to prevent users from editing the same frame at the same time.

Generality of representation. The database should be general in the sense that it can represent arbitrary textual and graphical information (though the current ZOG system supports only text). Thus, one can represent objects as varied as research articles, one-page letters, Pascal programs, and budgets.

Network structures. The database should have a network structure in which data items can be linked to other data items in the database.

Tree structures. Although the database can represent arbitrary network structures, there should be a strong preference for the representation of tree structures. This is largely a convention for the way frames are interconnected when built. However, the system should support a distinction between selections on a frame that point to lower levels in a tree versus selections that are "cross-references" to frames not within the tree structure. In ZOG, there is such a distinction: *options* are the "tree selections", and *local pads* are the "cross-references".

Menus. The use of menus of selectable items should be ubiquitous -- the database should contain nothing *but* menus. Menus may contain "pure" information items (like short paragraphs or points in an itemization) as well as items that lead to other frames or have associated actions. This allows graceful growth of the database, as pure information items are expanded to link to new frames

containing more detailed information.

Multi-level organization. The database should have the following four levels of organization:

- At the bottom level, there is substructure on single data items -- e.g., a name component or value component of a data item. (In ZOG this is supported only by embedded text conventions).
- One level up from the bottom there are single data items -- a multi-line unit of text (a phrase or paragraph), or a graphical entity (line or picture).
- One level up from single items are collections (menus) of items. This is the unit of display to the user (a frame in ZOG).
- At the top level, there are functional groupings of menus (called *subnets* in ZOG). The menus in the collection often have a common structure taken from a *schema* menu that is copied when they are created.

Default view. A default view of the data should be provided that organizes the tree structures in a breadth-first fashion, with a frame being a node in the tree. This view contains explicit positioning information that determines how the frame will be displayed to the user. Representation of the database on secondary storage is optimized for efficient access to the default view by storing all the items of one frame together. Other views of the data must be produced by processing the breadth-first representation.

2.3. User Interaction

The ZOG philosophy also makes a major commitment to a particular style of interaction between the user and the system, as indicated by the following principles:

Menu selection. Almost all interaction with the system should be done by making selections from the currently displayed menus. The exceptions are using the editor, plus the few cases where the user is prompted for a simple response, such as the identifier of a frame to be displayed. Even creation of new frames is triggered by selection of menu items that do not yet lead to other frames.

Fast response. Response of the system to a user selection should be fast. For the normal case where the selection simply results in the display of another frame, response should be well under one second. For standard video display terminals, this means that transmission speeds must be 9600 baud or better. The fast response requirement has strong implications for the implementation: when a frame is retrieved and displayed, there is no time to dynamically compose the display by gathering data from many locations in the database.

Browsing. The default mode of the system should be browsing through the network of linked menus. This distinguishes the system from most other database systems, which require that a query be formulated to access data.

Active selections. In addition to linking to other frames, selections should have *actions* associated with them, which are executed whenever the user chooses the selection. There is a set of common utility functions provided by the system, along with a simple language to express sequences of these functions as selection actions. One of the functions provided is to call arbitrary programs that have been integrated within the system (see the discussion of *functional extension* below).

No hidden selections. The currently visible functions should be the only functions available to the user. For example, there should be no hidden keyboard commands that a user has to remember. This means that users can rely totally on their recognition memory, i.e., their ability to recognize that a particular selection they see displayed will provide the desired function.[1]

Common commands. Some common commands are of such general use that they should be available on every frame. In the ZOG system these are called *global pads*. The ideal number of such command selections is probably around twenty. If there are fewer than twenty the user's efficiency may be impaired because of the extra time to access common functions; more than twenty and the frame becomes too cluttered and confusing to beginners.

Editor. There should be a general editor that operates on individual frames and is always available as one of the common command selections.

No scrolling. Frames should not be permitted to grow larger than the size of the available display space, and thus there should be no functions for scrolling the information within a single frame. This piece of philosophy is one that distinguishes ZOG from most other systems that provide similar functions. It is motivated largely by a desire for simplicity and by the need for all available functions to be visible. There are, of course, occasions where one must represent a linear data structure that is too large to fit on a single frame (though in our experience these are rare). In these cases, one can either introduce extra levels of hierarchy,[2] or simply link frames together in a linear sequence.

---

[1] The current ZOG system violates this bit of philosophy in several places, particularly in the editor. But an editor may require exceptions to this principle in general, because of the need for a high rate of interaction and a fairly wide and flat command structure for efficient expert use.

[2] One of our ZOG users coined the term "nodes of convenience" for these extra frames, which later turned into "nodes of inconvenience" in certain circumstances where they seemed particularly awkward.

2.4. Functional extension

A ZOG system needs more than just a database with an interface -- it needs some mechanism for extending the system to provide new functions for the user. The following principles describe how this is done:

Mapping data structures. The first step in adding a new application to the system should be to map the data structures of the new application into frame formats and interconnection structures within the database. Frames can be used as *record* structures, with individual items being fields in the record. The interconnection structure of frames can be used to represent hierarchical relationships, and to create access paths (indexes) to stored data.

Imbedded programs. Programs that are needed to implement new functions are written in a special way that allows them to be imbedded within the system, so that they can be used without having to leave the system. These *agents* can be invoked within the system via active menu selections.

Environment frames. Agents are invoked and controlled from special frames called *environment frames*, which contain slots for all the input parameters for the agent and slots for the agent to store output values.[3] There is a special editor (the *slot editor*) for environment frames that provides for efficient filling-in of the agent's input slots. The slot editor does type-checking on input values and allows values to be selected from a menu where appropriate. There is a special menu item on the environment frame that, when selected, causes the agent to begin execution.

Frames for input/output. Agents directly access frames in the database to get input data. (The environment frame would typically have an agent input slot that contains a pointer into the appropriate area of the database). When agents need to produce large data objects as output, they simply create frame structures in the database, which are then available to the user in permanent form.

3. Applications of ZOG

From the very beginning, our work on ZOG has often been driven by particular applications. We wouldn't wish it any other way, because we believe that real applications are the best breeding grounds for new research ideas. In the early days, we attempted to apply ZOG to our own professional needs -- for project management and for teaching ZOG to new users. In recent years, we

---

[3] This scheme is an adaptation of the environment mechanism developed for the CMU Cousin system (Hayes & Szekely, 1983).

have devoted most of our effort to developing the USS CARL VINSON application. Although it took us away from our normal research lives, it proved to be an excellent source of new discoveries about ZOG.

3.1. Some application areas we have explored

Below we list some general categories of applications we have worked on, along with a brief description of how ZOG has been used in each.

Database systems. One of the major functions that ZOG provides is storage and retrieval of information. ZOG can therefore be viewed as a database system that uses the "network" data model. On the storage side, ZOG allows users to grow the database one frame at a time, using any existing frame as a departure point (much like expanding a network of roads into the countryside). The databases that have been constructed using ZOG range from small personal databases to large project databases. Most of the frames in these databases were created directly by users, but in some cases agents were used to automate the frame-creation process. For instance, a frame development system called BROWSE partially automated the construction of a library database in ZOG frame form (Fox & Palay, 1979).[4] On the retrieval side, ZOG provides navigation commands (some of the global pads) which enable the user to traverse the network. These commands permit users to browse in an unfamilar database and progressively acquire a model of its contents and structure.

Management information systems. We have accumulated some meaty experience with ZOG as a management information system. For several years, we have been using a ZOG database to manage the ZOG project. This database contains all of our documents, software, reports, schedules and many other types of project-related information. The ZOG database for the CARL VINSON can also be viewed as a management information system, since its primary purpose is to help the Commanding Officer and his department heads administer the ship.

Instruction/training. ZOG has been used as a training system in several capacities: (1) providing on-line help; (2) as a guidance system for using other on-line systems by shielding users from the idiosyncracies of a particular interface; and (3) as an index and control mechanism for a videodisc player.

Document management. We have used ZOG as a document production environment by mapping

---

[4] BROWSE began with an already existing bibliography database separate from ZOG, and provided a way to automatically construct ZOG frames from the bibliography entries, along with indexing frames that provided access to the entries according to a standard classification scheme.

the natural hierarchical structure of documents into trees of ZOG frames. ZOG can then be used as a kind of tree-structured document editor. We also developed an agent that traverses the tree of frames and transforms the structure and content of those frames into a form suitable as input to a document formatting system (in this case Scribe). This provides a means for automatically producing high-quality hardcopy documents from ZOG frames, without having to explicitly provide most of the formatting commands.[5]

Software management. ZOG has also been used as a programming environment. The programmer represents code in frames, after which another special agent is applied to the frames to generate a compilable version of the code in a file. By equating the notion of a frame with the notion of a block in block-structured programming languages, ZOG provides a natural environment for a top-down, stagewise approach to developing code. Most of the code for the ZOG system itself is now in frames, along with various data structures associated with software development, such as change logs, bug reports, old versions, design notes, and user guides. Figure 3-1 shows a sample frame of Pascal code from the middle of an agent called *AgOld*. Note that each statement is a separate option, and that options 3 and 4 both lead to other frames (no "-" after the "."). Figure 3-2 shows the frame that option 4 leads to. Note that the BEGIN and END around the two statements are implicit -- they are added by the agent that writes the code to a file for compilation.

---

[5] This paper itself was created totally from within ZOG. As a result, its style is decidely more structured and "chunky" than is the normal case. Some of us believe this is a good thing; others may reasonably disagree.

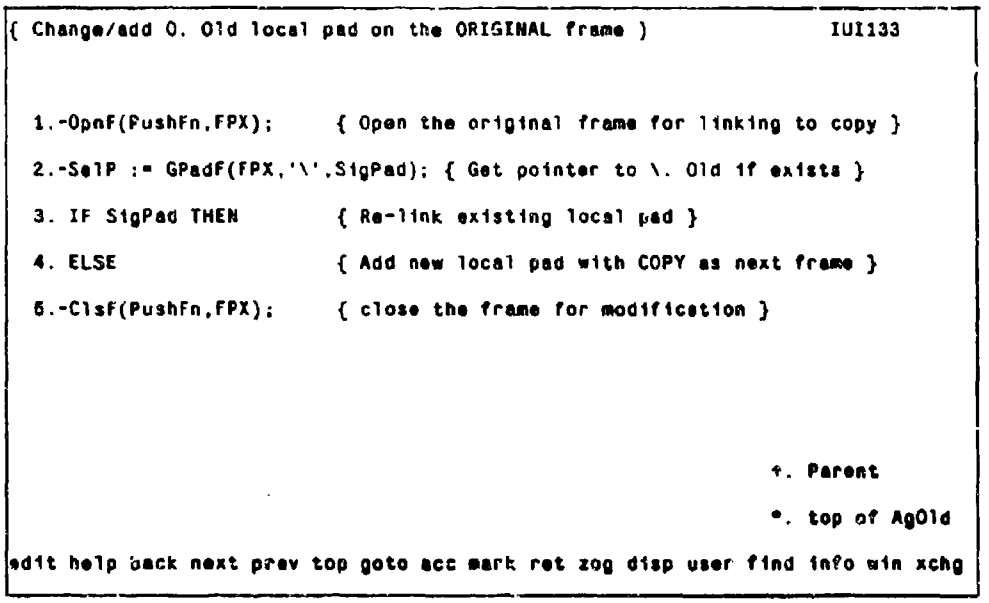


Figure 3-1: First example Pascal code frame

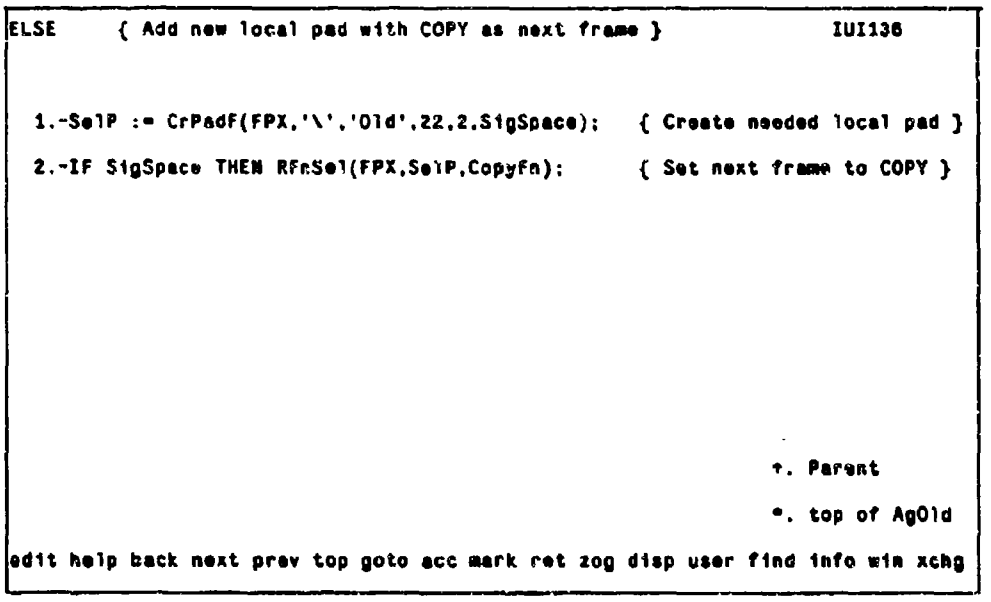


Figure 3-2: Second example Pascal code frame

Electronic communication. We have used ZOG for various forms of electronic communication such as electronic mail and bulletin boards. In ZOG, electronic communication occurs in a manner quite different from conventional electronic communication systems. "Mail" messages are placed on

ZOG frames according to topic, so that all messages on a certain topic can be viewed together. Each message does not have to re-establish context as it would if part of a series of unrelated messages. In other words, ZOG provides *logical coherence* for mail messages. Figure 3-3 shows an example of an actual interchange of several mail messages between two Navy officers ("cvmdf" and "cvmlr") over a five day period.

```
Mail: Flying time for mdf                                          IUI134

  1.-Mike: Please schedule max flying for Tuesday and Wednesday, and
     possibly Thursday afternoon.   [cvmdf 9/8/81]

  2.-It turns out the best bet is to go with VR56, get your name
     on a yellow sheet and spend the day flying on a C-9 going
     from place to place. I may join you on one of these expeditions
     and we can spend the whole day working "in the air". [cvmlr 9/8/81]

  3.-Sounds great...will you please make arrangements with VR56? [cvmdf 9/8/81]

  4.-Scratch option 2 its only for flight surgeons.  It looks like
     VRC40 is now best bet.  There's no need to schedule in advance
     just show up.[cvmlr 9/9/81]

  5.-Mark, I just remembered, I will be in Maneuvering Board School
     next week.[cvmlr 9/9/81]

  6.-I will also be working an option at VA42 to go flying in the afternoons
     at Oceana after school. We'll see if we can't work something out where
     I pick you up and take you too. [cvmlr 9/12/81]  [cvmdf 9/12/81]

edit help back next prev top goto acc mark ret zog disp user find info win xchg
```

Figure 3-3: Mail frame example

3.2. The ZOG/ USS CARL VINSON project

As mentioned in the introductory section, the ZOG/ USS CARL VINSON project was a joint project between the ZOG researchers at CMU and the Captain and crew of the USS CARL VINSON. The project officially began in early 1980. In early 1983, a complete ZOG-based application system was installed on board the ship, running on 28 PERQ computers connected via an Ethernet network. The system provided a transparent, distributed database of ZOG frames (initially over 20,000 frames), with access times of about 0.6 seconds for frames residing on the local disk, and 1.2 seconds for frames on remote machines. There were over 30 agents (application programs) that provided, in conjunction with the basic ZOG system, four main application functions:

Ship Organization and Regulation Manual. One of the original project goals was to represent the Ship Organization and Regulation Manual (SORM) in a ZOG net, so that its contents could be accessed on-line. The CARL VINSON's SORM had not yet been written. Thus ZOG was used as a

document production environment for developing the SORM. A major objective for the SORM was to make its contents usable not only by human users, but also by agents (management application programs).

Interactive task management system. Since the *responsibilities* section of the SORM is quite structured, agents can be used to support task management. In simple terms, tasks described in the SORM are copied and their generic portions are instantiated to reflect the particulars at hand (times, people, etc.). These specific structures are then used to track the status of these tasks and generate reports of various types. An example task is "getting underway"; this task resembles a space launch countdown, with hundreds of synchronized subtasks occurring over a three day period.

Technical manuals. A function that was added to the original project agenda was the development of on-line technical manuals for the aircraft and weapons elevators. Here, as with the SORM, new manuals were needed. The manuals were produced within ZOG by personnel from the shipbuilding company, as well as members of the crew. In addition, the on-line version of the manual was integrated with videodisk material. Embedded in relevant places throughout the on-line manual are ZOG selections that control an adjacent videodisk player to play specific portions of the disk.

User interface for AirPlan. Another function not in the original plan was an interface to an expert artificial intelligence system called *AirPlan*.[6] AirPlan assists the Air Operations department in the launch and recovery of the ship's aircraft. One of AirPlan's functions is to alert decision makers when currently available options are soon to disappear (for example, that a given airborne plane will go below the requisite fuel for flying to an alternate landing site). ZOG provides an interface to AirPlan for both input and output. The slot editor provides an efficient means for updating AirPlan on the state of the world. Output from AirPlan is sent to ZOG and displayed in frames that any workstation on the network may access. An incremental display feature was added to ZOG so that incremental changes to these output frames are highlighted; this reduces the cognitive overhead of identifying what information on the display has actually changed.

4. A Critique of ZOG

In our long experience with ZOG, we have been on the lookout for strengths and weaknesses that have appeared as we have pushed ZOG in new directions. In the following two sections we present the major strong and weak points of ZOG, each with a brief elaboration. A third section summarizes the sources of evidence that support our analysis of strong and weak points.

---

[6] The expert system itself was developed by another group of researchers at CMU, headed by John McDermott.

4.1. ZOG's strong points

We believe that ZOG has many strong points; in fact, it has often surprised us (even with our understandable optimism) how well ZOG has adapted to the many demands we have made of it. The most important of the strong points are described below:

Robust enough to be put into operational use. Compared with other types of research interfaces, such as natural language interfaces, ZOG is extremely robust. This is because of its simplicity and its openness. Many new tasks can still be performed, though perhaps awkwardly at first, even if they lie outside the capabilities of existing ZOG agents.

Easy for computer novices to learn and use. Computer novices can learn to do basic ZOG navigation in under a half hour. Learning enough of the ZOG editor to add new material in a reasonably effective manner takes about another two hours.

Users do not outgrow ZOG as they become expert. As users gain experience, they seem to become progressively more attached to ZOG. Several aspects of ZOG seem to account for this phenomenon: (1) the system's high response rate makes menu selection competitive with a command language interface; (2) experts tend to use a much broader range of functions than novices, and thus appreciate the fact that ZOG eases their memory burden; and (3) the capacity of the database to absorb multiple indices allows experts to develop additional structures that make their use more efficient.

Can assimilate and integrate many different applications. ZOG has demonstrated that it is capable of assimilating and integrating a wide range of applications. We believe this is primarily due to the generality of ZOG frame structures for representing information of all kinds.

Supports database browsing. ZOG provides good support for browsing because of its rapid response and the network structure of its database. The ability to search a database by query methods is strongly dependent on having a model of what information the system contains. Browsing provides a means for a user to start searching with minimal *a priori* knowledge, yet the user can exploit what he learns about the database over time. In ZOG, the user's knowledge of the contents and structure of the database is continuously reinforced through use.

Supports large databases. The structure of the database is such that there is no limit to the size ZOG can operate with, except for those limits inherited from the hardware or operating system levels. Consequently, the maximum size of the database is a function of the collective size of secondary

memory. In a distributed system with many nodes this size can be extremely large, but then the bandwidth of the network may begin to degrade system response.

Can exploit schemas for building databases. A *schema* is essentially a chunk of ZOG data (e.g., a single frame or a tree of frames) that contains variable parts; a schema can be *instantiated* by copying it and then providing constant values for the variable parts of the copy. Schemas aid in the building of a number of data objects that have some common parts. In ZOG, a simple schema mechanism is provided through the ability to copy a schema frame when a new frame is created, or to copy whole trees of frames. Ramakrishna (1981) developed more elaborate schema mechanisms for ZOG and studied their use experimentally.

Can be used as the sole interface (shell) for an operating system. ZOG has the capability to assimilate all the utility functions normally associated with an operating system shell. Single selection actions can invoke simple utilities, and environment frames can be used for utilities with several parameters (essentially revising the utilities to behave as ZOG agents).

Has a simplified model of window use. In the PERQ version of ZOG, there are three windows (two for frames and one for messages), fixed in size and in location. The need to maintain multiple active working contexts, which is often solved by allowing arbitrary numbers of overlapping windows, is satisfied in ZOG by navigating among the different contexts in a single ZOG window. This simplified model may have some real virtues such as ease of learnability.

Has a simplified model of multiple processes. There is only a *single* ZOG process that interacts directly with the user. All other processes are simply background processes running agents, which operate on the ZOG database but do not need to communicate directly with the user. This is a simpler model than the common one where each process has its own display window, and can potentially interact with the user independently of the other processes.

Can be used as an interface mechanism for video discs. Video discs are wonderful devices for storing visual material, but they lack any structured way of gaining access to the stored images -- there is simply a flat sequence of tens of thousands of images. ZOG frames can be built to provide structured access to the material, with special actions on ZOG frames that can call up particular images or motion scenes.

Can make good use of a pointing device (mouse). Because of the high proportion of selection

operations by the user, ZOG is well suited for using a pointing device such as a mouse.[7] This is especially true of ZOG's editor since positioning the cursor and repositioning items are frequent operations.

Can exploit distributed systems. We have come to appreciate ZOG's ability to exploit the positive features of distributed systems while shielding the additional complexity from the user. ZOG's implementation of the distributed database allows a user to think in terms of a large, single database and not worry about which machine particular frames are on. Furthermore, the user need not be aware which machine on the network is actually executing an agent the user invoked.

Supports a community of communicating users. A large, shared database provides a "commons" in which users can jointly develop and share data, rather than simply exchange it. This is important to the individual user because his ability to assimilate information far outstrips his ability to generate it. Large, interesting databases can only be built by many hands.

4.2. ZOG's weak points

Below are the weaknesses of ZOG that we have become aware of:

ZOG sacrifices efficiency of particular applications to get integration. Because ZOG is attempting to be all things to a user, compromises must be made when fitting an application into the ZOG structure. The ZOG data structures for an application are usually less efficient in both time to process and space than they would be for an application system built for that special function. Also, accomplishing some application functions may take more steps in ZOG than in a specialized implementation, because more general, and hence less powerful, operators are being used.

ZOG does not support a fast database query language. The ZOG database is represented on secondary storage as text files for flexibility -- parts of the database can be easily backed up or moved from one system to another, and standard text editors can be used on the database in emergency situations. There is thus a substantial overhead incurred in "parsing" and "unparsing" frames for internal system use.

Inexperienced users can get lost. When a user faces a large, complex ZOG database that he is unfamiliar with, there is a strong possibility that he may become disoriented. (See Mantei (1982) for a

---

[7] We explored the use of touch screens, but learned that users found them inaccurate and inconvenient. In our opinion, the mouse is a clear winner over all other positioning devices for use with ZOG.

study of disorientation in a ZOG database specially constructed for experimental purposes). However, it has been our experience that this problem is not nearly as severe as we once feared, partly because ZOG has adequate means to find your way again once you have become lost.

Biased too much toward breadth-first view. ZOG frames store information in a "breadth-first" fashion, and thus the user is forced to operate exclusively with this form. There are, however, times when other views of the data (e.g., depth-first, as with typical hardcopy) would be more appropriate to the user's momentary needs.

ZOG depends too critically on the speed of the disk technology. ZOG needs to access a few thousand bytes from secondary storage within every response cycle. This puts a stringent requirement on the access time of the disk hardware and the overhead of the file system software. Flexible disk ("floppy") drives are too slow; inexpensive hard disks are barely adequate.

Can't represent all states of a complex, unordered task environment. A complex, unordered task is the proper domain for a rule-based system, where the individual rules can freely apply in any order as the current situation dictates. Attempting to create a decision tree with ZOG frames to analyze some dynamic situation runs into difficulty because of the unpredictable order of state changes, causing a combinatorial explosion in the size of the requisite ZOG structure.

Can't handle highly dynamic data. ZOG cannot cope with situations where many updates to the database are required every second, particularly if the updates are widely spread throughout the database. This is due to the fact that data is updated in frame-sized chunks, rather than in smaller units, and the overheads in reading and then rewriting a frame are substantial -- a second or two on the PERQ.

Lack of graphics and multiple fonts. ZOG was originally developed for terminals that did not support graphics and multiple fonts. When we moved ZOG to PERQs for the CARL VINSON, we took the conservative route of emulating character terminals rather than exploiting the PERQ graphics capability. There is no fundamental reason why these capabilities could not be added to ZOG.

ZOG cannot be used over standard telecommunication lines. Transmission rates of 1200 baud, which are the norm for dialup phone connections, are simply too slow for proper use of ZOG -- experts become impatient at those speeds. Even 9600 baud is barely adequate for experts.

4.3. Evidence for our beliefs

The strong and weak points listed above are of course not simple facts -- they are merely our beliefs. However, they are the product of a substantial body of experience with ZOG, both in the university environment and on board the USS CARL VINSON. Below, we discuss four different aspects of this experience: our own use of ZOG, use by the Navy, laboratory experiments, and instrumentation of the system.

Our own use of ZOG. Our primary source of beliefs about ZOG is our own use of the system over the past six years. The early years saw rapid iteration on the ZOG design, with relatively little serious use of the system for applications. Then during 1978 and 1979 we made a serious attempt to coordinate the management of the ZOG Project (involving about 10 people altogether) by using ZOG to hold status information, discussion of issues, meeting notices, and other information of general interest to project members. When we began the CARL VINSON project in 1980, we again attempted to use ZOG for project communication, but this time with the added twist that some of the project members were located in Newport News, VA. We were more successful this time, partly because the remoteness of some project members made other means of communication more difficult, and partly because we had more computing cycles available and faster communication lines. During the course of the VINSON project, we expanded our use of ZOG into document and software management, and actually reached a point where several of us used ZOG for nearly everything we did throughout the workday. For the three-year period beginning in October, 1980, there were over 200,000 sessions of ZOG use on our VAX version -- and by the middle of 1982, most ZOG use had shifted from the VAX to PERQs, for which we don't have session counts.

Use on board the CARL VINSON. Use of ZOG by members of the CARL VINSON crew began in late 1980 with the use of a VAX at CMU over leased phone lines. This usage continued fairly steadily for two years, with six terminal lines available for most of that period. The main activity, besides project communication, was the building of databases that the ship would use during cruise -- the SORM and the two elevator technical manuals. More than 20,000 frames were created on the ZOG VAX for later transfer to the ship's PERQ system. In early 1982, the ship used a prototype version of the management application software on the ZOG VAX to produce several large management schedules for the ship's operation, such as the Builders' Trials. Then, beginning in March, 1983, the ship had the completed ZOG system available, although there were still many hardware and software problems being ironed out. We know that there were at least 500 sessions of ZOG use on board during April and May, but most of the usage was limited to a small subset of the total of 28 PERQs. By the end of September, usage had increased considerably, with about 30 serious users and involvement by 85-90% of the ship's departments.

Laboratory experiments. From the beginning, we have viewed ZOG as a vehicle for research in the psychology of human-computer interaction. We created a special laboratory where we could do controlled experiments of people using ZOG. One of our main interests has been to study how people use ZOG to learn how to use ZOG, so we do have some basis for beliefs about ZOG being easy to learn. We have also studied use of the ZOG editor extensively. One specific experiment evaluated the speed of expert use of the ZOG editor, with the result that ZOG compared unfavorably with several other commonly used editors, such as a Wang word processor (Robertson, C., McCracken & Newell, 1981).[8] Another specific study compared the time to learn the ZOG editor against data on eight other editors, and showed that ZOG's editor is in the middle of the range of ease of learnability (Robertson & Akscyn, 1982). This study also compared different teaching tools: a human teacher, an on-line tutorial, an on-line manual, and an off-line manual; results indicated that on-line and off-line manual users take about the same time to complete a standard instruction sequence, but off-line manual users use the editor more effectively at the end of the sequence.

System instrumentation. Both the VAX and PERQ versions of ZOG have been instrumented to save statistics for each session of use. These statistics summarize the activity of the session with information such as number of frames viewed, number of frames edited, distributions of times spent viewing each frame and editing each frame, which subnets were visited, and so on. Originally we planned to use the data to instruct the iterative development of the system; however, on the CARL VINSON the data is also being used for an official Navy evaluation of ZOG use on board the ship. Although we have done some spot analysis of the statistics, a systematic attack on this great mass of data still lies ahead.

5. The Future of ZOG

There is now work going on outside CMU to create a follow-on to ZOG called KMS (Knowledge Management System). KMS is designed to remedy many of the shortcomings of the ZOG implementations, and specifically to make full use of the potential of high-resolution display technology. Some of the features of KMS that represent improvements over ZOG are the following:

- Graphics (lines, rectangles, curves, picture images) and multiple fonts.
- Greater use of the pointing device to specify objects and parts of objects to operate on, and screen cursor images to provide feedback of the current system context.
- Direct output of good quality hardcopy (no need for a separate formatting system that

[8] We do believe that ZOG's editor is due for replacement, based as it is on old editor techology that is clearly inferior to the new breed of screen editors.

operates as a post-processor).

- Copying material easily across frame boundaries.
- Additional intrinsic views of the database -- specifically, a depth-first view.
- Closer integration of the editor with the rest of system, to make editing seem more natural, rather than a special mode that one is continually entering and leaving.

Despite the years of work on ZOG, we still see many rich possibilities waiting to be explored. We have only scratched the surface of some of the ZOG applications we have worked on, and there are many potential applications as yet untouched. We are particularly interested in advancing the use of ZOG as a high-quality documentation environment, and as a programming environment, since these are tools that can greatly enhance our own daily work.

6. Acknowledgements

We wish to acknowledge the contributions of many people over the years. Those who have been involved with ZOG at CMU: Allen Newell, George Robertson, Kamila Robertson, Elise Yoder, Sandy Esch, Patty Nazarek, Angela Gugliotta, Marilyn Mantei, Kamesh Ramakrishna, Roy Taylor, Mark Fox, and Andy Palay. Those officers from the USS CARL VINSON who worked with us at CMU: Mark Frost, Paul Fischbeck, Hal Powell, Russ Shoop, and Rich Anderson. Captain Richard Martin, Captain Tom Mercer, and other officers and crew of the USS CARL VINSON. And finally, Marvin Denicoff from the Office of Naval Research, our original ZOG research sponsor. We would also like to thank Elise Yoder for her extensive comments on this paper.

This work was supported by the Office of Naval Research under contract N00014-76-0874. It was also partially supported by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under contract F33615-78-C-1551. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research, the Defense Advanced Research Projects Agency, or the U.S. Government.

7. References

Carroll, J.M. (1982). The adventure of getting to know a computer. *IEEE Computer*, pp. 49-58 (November).

Fox, M. & Palay, A. (1979). The BROWSE system: an introduction. *Proceedings of the Annual Conference of the American Society of Information Science*, Minneapolis (October), pp.183-193.

Hayes, P. & Szekely, P. (1983). Graceful interaction through the COUSIN user interface. *International Journal of Man-Mahine Studies*, **19**, pp. 285-305.

Mantei, M. (1982). *A Study of Disorientation Behavior in ZOG*, PhD thesis, University of Southern California.

Newell, A., McCracken, D., Robertson, G. & Akscyn, R. (1981). ZOG and the USS CARL VINSON, *Computer Science Research Review, Carnegie-Mellon University*, pp. 95-118.

Ramakrishna, K. (1981). *Schematization as an Aid to Organizing ZOG Information Nets*, PhD thesis, Computer Science Department, Carnegie-Mellon University.

Robertson, C.K. & Akscyn, R. (1982). Experimental evaluation of tools for teaching the ZOG frame editor. *Proceedings of the International Conference on Man/Machine Systems*, Manchester, UK (July).

Robertson, C.K., McCracken, D. & Newell, A. (1981). Experimental evaluation of the ZOG frame editor. *Proceedings of the 7th Canadian Man-Computer Communications Conference*, Waterloo, Ontario (June), pp. 115-123.

Robertson, G., McCracken, D. & Newell, A. (1981). The ZOG approach to man-machine communication. *International Journal of Man-Machine Studies*, **14**, pp. 461-488.

Schultz, J. & Davis, L. (1979). The technology of PROMIS. *Proceedings of the IEEE*, 67, (September), pp. 1237-1244.

Shneiderman, B. (1983). Direct manipulation: a step beyond programming languages. *IEEE Computer*, (August), pp. 57-69.